OptiGuide: An Efficient Domain-Independent Package Recommender System Based on Multi-Objective Optimization and User Decision Guidance

Tahani Almanie
Computer Science Department
George Mason University, and
Information Systems Department
King Saud University
Talmanie@gmu.edu

Alexander Brodsky
Computer Science Department
George Mason University
brodsky@gmu.edu

Abstract

While traditional recommender systems focus on single items, there is an emerging demand for package recommenders that can suggest composite items based on multiple criteria. For instance, they can recommend a combination of dishes based on price, cuisine, and dietary restrictions. Several challenges arise when dealing with package recommenders, including the complexity of the decision-making process and the need of handling trade-offs among conflicting objectives. We introduce OptiGuide, a domain-independent package recommender system that uses efficient multi-objective optimization techniques to guide users effectively in finding their Pareto-optimal recommendations. The user is engaged in the decision-making by capturing their user-system interactions and offering customization options to help them find their optimal recommendation. The system employs preprocessing algorithms to balance the need for quick response times with the computational complexity of the optimization process. A dynamic configuration mechanism is adopted using a pluggable analytic model to enable system versatility across diverse domains.

Keywords: Package Recommender, Multi-Objective, Decision Guidance, Optimization, Pareto-Optimal.

1. Introduction

Recommender systems play a vital role in diverse domains and applications by offering personalized recommendations tailored to individual preferences (Fayyaz et al., 2020). While traditional recommenders focus on single products or services, there is an emerging demand for package recommender systems that can suggest composite or bundled products and services based on multiple criteria or objectives (Deng et al., 2012). For instance, they can recommend a composite of travel-related options such as destinations, accommodations, and transportation based on traveler preferences, budget, and duration of stay.

Several challenges arise when dealing with package recommender systems, one of which is the complexity of the decision-making process. Due to the existence of multiple objectives and a vast range of alternatives, mathematical optimization is required to generate optimal recommendations. Moreover, package recommenders must be able to handle trade-offs among multiple conflicting objectives, such as prioritizing price over quality or vice versa. User guidance is required to obtain more reliable evaluations, especially when dealing with conflicting objectives or trade-offs (Zheng & Wang, 2022).

Furthermore, the majority of existing recommender systems provide recommendations for items within a single specific domain (Dacrema et al., 2022). This domain-dependence poses a critical limitation as it restricts their adaptability and effectiveness when applied to different domains. For instance, a recommender system designed for movie suggestions might not perform effectively when used for recommending travel destinations.

Therefore, there is a need for a package recommender system that is domain-independent, employs multi-objective optimization, and provides user decision guidance. Extensive research has been carried out in recent years to address these challenges.

Zhang et al. propose a multi-objective optimization approach for food recommendation that considers four objectives: user preference, user diet pattern, food nutritional values and food diversity and applies Pareto-based algorithms to find the optimal solutions (Zhang et al., 2022). MoParkeR is a multi-objective parking recommendation system that uses a non-dominated sorting technique to find Pareto-optimal parking spots based on factors such as fare, walking distance, and travel time (Rahaman et al., 2021). While both systems apply multi-objective optimization through the utilization of Pareto optimality, they are domain-specific and do not address package recommendations or user decision guidance.

URI: https://hdl.handle.net/10125/106570
978-0-9981331-7-1

Wibowo et al. propose a method for recommending clothing packages by combining matrix factorization for collaborative filtering with fashion criteria such as color, formality, and patterns. Focus groups were conducted to improve the algorithm's performance (Wibowo et al., 2018). Another work presents a graph-based approach to recommend personalized travel packages that contain a sequence of multiple points of interest based on user preferences, budget, and spatio-temporal constraints (Hti & Desarkar, 2018). However, they do not provide a user feedback mechanism or interactive interface for refining preferences in the recommended packages. Despite using package recommendations for multiple objectives, neither of the two methods is domain-independent or addresses the need for user decision guidance.

MovieBrain is a Google Chrome extension that integrates IMDb's movie recommendation system. It provides a personalized movie recommendation experience by offering interactive settings and filters based on factors such as movie genres and rating scores (Dooms et al., 2014). Another research proposes a recommender system that puts some control in the hands of users by allowing them to adjust the weight of non-personalized factors such as the popularity and recency of movies (Harper et al., 2015). Both systems incorporate user decision guidance and consider multiple criteria; however, they are domain-specific and do not provide package recommendations.

MoFIR is a fairness-aware recommendation system that uses multi-objective reinforcement learning to find an optimal trade-off between fairness and utility. The system seeks a Pareto efficient solution, which can be applied to different recommendation domains. MoFIR enables decision-makers to specify preferences for fairness and utility, guiding them in choosing recommendations that align with their business needs (Ge et al., 2022). However, the system is not designed for package recommendations.

Kouris et al. present a framework to recommend packages that best fit users' preferences while satisfying their constraints. The framework employs both an optimal algorithm, combining collaborative filtering with a graph model, and a computationally efficient greedy algorithm (Kouris et al., 2020). This system can be adapted to various domains; however, it does not engage users in the decision-making process.

CAPORS-IUX provided a valuable solution for the mentioned challenges by introducing a methodology and presenting a system for generating composite alternative (package) recommendations based on Pareto-optimal trade-offs and extracting the utility of an individual user (Jeffries & Brodsky, 2018). The system employs multi-objective optimization with arbitrary metrics, captures user-system interactions, and conducts re-optimizations to iteratively find a feasible recommendation closest to the user's utility. However, CAPORS-IUX requires multi-objective optimization in response to user's actions, which can be impractical to perform in real-time, especially for more complex underlying domain models, which are needed for recommendations in areas such as supply chain, infrastructure investment or electric grid. The user interface also has certain limitations including navigation across the complex recommendation and comparing different alternatives. Moreover, while CAPORS-IUX is capable of generating recommendations using arbitrary objectives, it requires further improvements to ensure better domain-independence functionality.

Addressing these limitations is exactly the focus of this paper. We introduce *OptiGuide*, a domain-independent package recommender system that extends the functionality of CAPORS-IUX. Most importantly, *OptiGuide* allows real-time responses to user's actions – even for very involved underlying models - due to preprocessing algorithms. More specifically, the contributions of this paper are as follows:

First and most important, we develop preprocessing algorithms that compute a database representing the Pareto front of the recommendation space, along with estimated user utility functions corresponding to each Pareto-optimal alternative in the database. Based on the preprocessed database, we develop efficient algorithms to provide users with real-time feedback to their actions, such as selecting the best option out of two-dimensional projections of the Pareto front, suggesting improvement for a particular objective, or re-estimation of the user's utility function. We use the Decision Guidance Analytic Language (DGAL) and Management System (DGMS) for executing optimizations (Brodsky & Luo, 2015, Nachawati et al., 2017), which supports pluggable predictive models expressed in Python, yet optimization based on the best available mathematical programming solvers.

Second, to enhance the user interface and provide a more satisfying user experience, we focus on several aspects such as the clarity of presented information, the level of customization offered to the user, and the ease of navigation across the complex recommendation.

Third, to make the system more generic and applicable across different domains, we implement a dynamic configuration mechanism that enables the system to adapt to various domains without requiring extensive changes to the underlying code. A configuration file is provided along with a pluggable analytic model where the domain-specific settings can be easily adjusted. Moreover, we allow the system to accept any number of objectives without limiting the number of generated recommendations as well.

The rest of paper is organized as follows. Section 2 describes the system using a supply chain sourcing example. Section 3 demonstrates the system architecture. Section 4 presents the details of the system implementation. Finally, Section 5 concludes this research and briefly provides directions for future work.

2. System description

We explain the functionality of the *OptiGuide* system along with the applied methodology through the use of a supply chain sourcing example (Jeffries & Brodsky, 2017).

Consider an example of a small supply chain having items such as tables, chairs, and cabinets, along with a demand for each item, and a set of suppliers. In this case, a recommendation is a set of orders, where an order contains a set of item quantities to be purchased from a particular supplier. The order must meet the demand constraint and is optimized according to one or more criteria, such as cost, carbon emissions, and manufacturing time. The target is to generate recommendations with the highest predicted utility to the user, where learning utility is a way of quantifying how much a user would prioritize an objective over another.

For generating the optimal recommended order, we adopt the methodology used by CAPORS-IUX which is captured as a state-activity diagram, shown in Figure 1 (Jeffries & Brodsky, 2018).

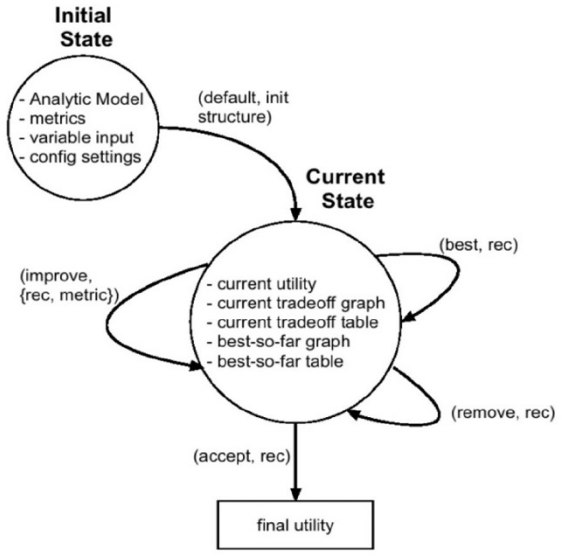


Figure 1. State diagram.

In order to initialize the recommender system for the given example, three domain-specific components need to be provided by the user: an analytic model, a variable input, and a set of configuration settings.

The analytic model formally describes the objectives and feasibility constraints as a function of parameters and control variables (Jeffries & Brodsky, 2018). Figure 2 shows the analytic model for the supply chain sourcing example. The variable input is used for exploring the feasible space of the recommendation alternatives, as shown in Figure 3. The configuration settings involve the objective (metric) definitions, the default objective for the user to consider, whether an objective is a minimization or a maximization metric, and the lower and upper bounds of each objective.

This analytic model takes as input the variable input file, which contains the demand for items *demand* and the purchase information *purchaseInfo* variables. For each pair of suppliers and items, the purchase information involves the price per unit *ppu*, carbon emissions per unit *co2pu*, manufacturing time per unit *manufTimePu*, available items *available*, along with the control variables quantities *qty*. The model defines how to compute each of the objectives, such as *cost*, carbon emissions *co2*, and manufacturing time *manufTime* based on the given input data. Additionally, it defines the feasibility constraints as the *nonNegQtysConstraint* to confirm non-negative quantities, the *availabilityConstraint* to assure that quantities do not exceed suppliers' availability limits, and the *demandSatisfiedConstraint* to ensure that the supply of items meets the demand. The model returns the computed objectives and constraints.

let
in = variableInput
sup = in(suppliers), item = in(items)
then

$$cost = \sum_{s \in sup} \sum_{i \in item} in(qty(s,i)) * in(ppu(s,i))$$

$$co2 = \sum_{s \in sup} \sum_{i \in item} in(qty(s,i)) * in(co2pu(s,i))$$

$$manufTime = \sum_{s \in sup} \sum_{i \in item} in(qty(s,i)) * in(manufTimePu(s,i))$$

$$nonNegQtysConstraint = \forall s \in sup, \forall i \in item \; in(qty(s,i)) \geq 0$$

$$availabilityConstraint = \forall s \in sup, \forall i \in item \; in(qty(s,i)) \leq in(available(s,i))$$

$$demandSatisfiedConstraint = \forall i \in item \; \sum_{s \in sup} in(qty(s,i)) \geq in(demand(i))$$

$$constraints = nonNegQtysConstraint \wedge availabilityConstraint \wedge demandSatisfiedConstraint$$

return *{cost, co2, manufTime, constraints}*

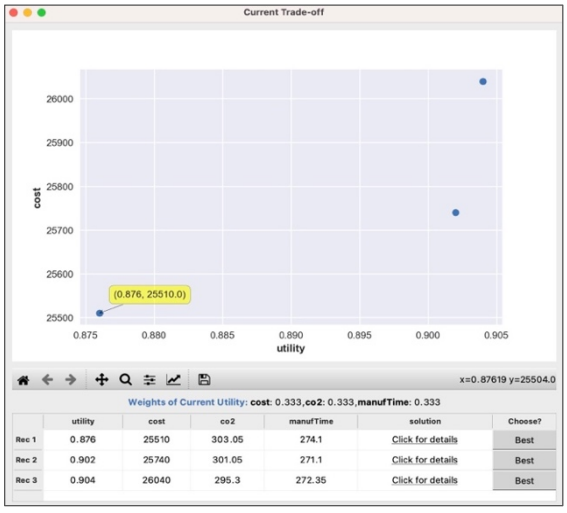
Figure 2. Analytic model.

```
{
  "demand": {"cabinet": 34,"chair": 140,"table": 75},

  "purchaseInfo": {

    "ppu": {"supplier1": {"cabinet": 170,"chair": 85,"table": 90},
            "supplier2": {"cabinet": 220,"chair": 62,"table": 150}},

    "co2pu": {"supplier1": {"cabinet": 2.7,"chair": 0.7,"table": 1.9},
              "supplier2": {"cabinet": 3.2,"chair": 0.9,"table": 0.75}},

    "manufTimePu": {"supplier1": {"cabinet": 2.2, "chair": 0.5,"table": 1.25},
                    "supplier2": {"cabinet": 1.9, "chair": 0.8, "table": 1.5}},

    "available": {"supplier1": {"cabinet": 25,"chair": 60,"table": 30},
                  "supplier2": {"cabinet": 15,"chair": 90,"table": 50}},

    "qty": {"supplier1": {"cabinet": {"dgalType": "int?"},
                          "chair": {"dgalType": "int?"},
                          "table": {"dgalType": "int?"}},
            "supplier2": {"cabinet": {"dgalType": "int?"},
                          "chair": {"dgalType": "int?"},
                          "table": {"dgalType": "int?"}}}
  }
}
```

Figure 3. Variable input.

The user interface contains four main sections: (1) Current Trade-off Graph: This graph displays all the generated recommendations aligned with a Pareto-optimal curve, where the x-axis shows the current utility, and the y-axis shows the objective to consider. (2) Current Trade-off Table: This table shows the information of each of the current recommendations including its computed objectives, the current utility value, and the pertained solution, along with a button for choosing the recommendation as "Best". (3) Best-So-Far Graph: A histogram graph that shows the normalized objectives of each best recommendation chosen by the user. 4) Best-So-Far Table: A table that lists the related information of all the best recommendations so far.

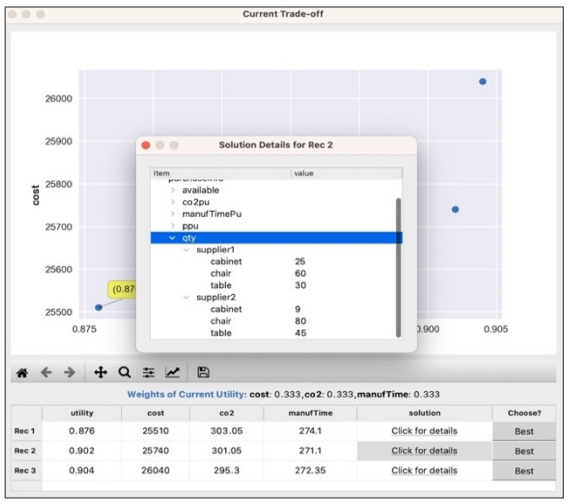
Initially, the current utility is calculated by assigning equal weights to each objective. The default objective is determined by the configuration setting and for this example, the y-axis represents the cost of ordering. The system generates an initial set of recommendations for the user to consider which fall along a Pareto-optimal curve, as shown in Figure 4.



	utility	cost	co2	manufTime	solution	Choose?
Rec 1	0.876	25510	303.05	274.1	Click for details	Best
Rec 2	0.902	25740	301.05	271.1	Click for details	Best
Rec 3	0.904	26040	295.3	272.35	Click for details	Best

Figure 4. Current trade-off graph and table.

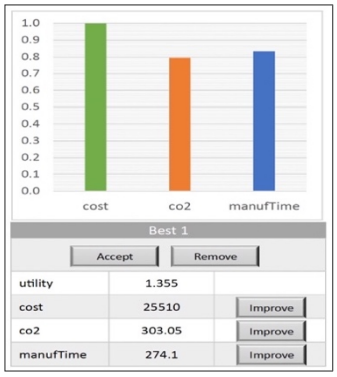
The user is provided with several display options for the graph to enhance their experience including the ability to zoom in on the graph for a closer exploration, pinpoint the precise objective values, and save the graph for future reference. In the Current Trade-off Table, the user has the option to view the details of a recommendation's solutions by clicking on "click for details." This action will open a window that displays the solution in a hierarchical view, as illustrated in Figure 5. In this example, the solution provides specific details, including the optimal quantities of items to be purchased from each supplier.



	utility	cost	co2	manufTime	solution	Choose?
Rec 1	0.876	25510	303.05	274.1	Click for details	Best
Rec 2	0.902	25740	301.05	271.1	Click for details	Best
Rec 3	0.904	26040	295.3	272.35	Click for details	Best

Figure 5. Solution details window.

Now, if the user chooses the first recommendation (Rec 1) as the best choice, this recommendation is added to the Best-So-Far Graph and Table, as shown in Figure 6. Besides, the system recalculates the current utility based on the selected recommendation, and the Current Trade-off Graph and Table are updated to reflect the new computed utility.



utility	1.355	
cost	25510	Improve
co2	303.05	Improve
manufTime	274.1	Improve

Figure 6. Best-so-far graph and table.

Next, the user is provided with three options in the Best-So-Far Table: "Improve" which generates a new set of recommendations by improving one of the objectives, " Remove" the recommendation from the list, or "Accept" the recommendation as the final optimal recommendation.

In our example, if the user wants to improve the recommendations based on carbon emissions (CO2), the "Improve" button is clicked and the Current Trade-off Graph and Table are updated with a new set of Pareto curve recommendations. As observed in Figure 7, the y-axis of the Current Trade-off Graph becomes the chosen objective to improve, and the x-axis is the last updated current utility. This process continues until the user decides on the optimal recommendation and selects the "Accept" button.

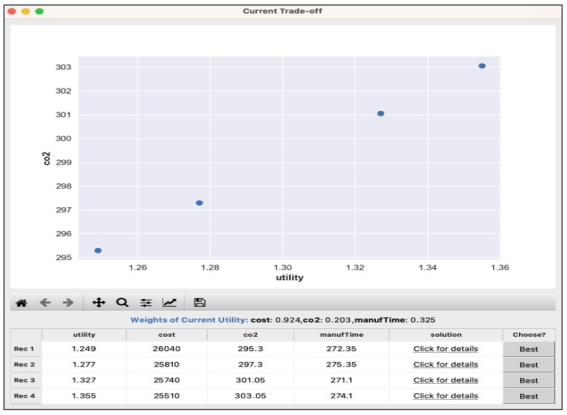


Figure 7. Updated current trade-off.

To further demonstrate the domain-independence of OptiGuide, consider using the system in the electric grid domain. The goal is to recommend the optimal allocation of power sources and transmission lines according to multiple objectives, such as minimizing the power generation cost, maximizing grid reliability, and reducing carbon emissions. Besides, the optimization process should satisfy specific constraints, including electricity demand, budget, and transmission capacity constraints. To initialize the recommender system for the given example, three domain-specific components should be provided: a variable input, an analytic model, and the configuration settings.

The variable input specifies the control variables and defines parameters, such as the available power sources, cost and carbon emissions per unit, reliability data for each source, transmission line capacities, and expected electricity demand. The analytic model defines how to compute each of the objectives and constraints based on the given input data. The configuration settings identify objective specifications such as the objective type and lower and upper bounds.

3. System architecture

The *OptiGuide* system is composed of two core internal components: The Recommendation Engine, which implements the preprocessing phase, and the Recommendation User Interface, which implements the runtime phase.

The Recommendation Engine must be initialized with a domain-specific data structure that contains: an analytic model, a variable input, and a set of configuration settings. The Recommendation Engine is further integrated with Unity DGMS to generate the domain specific recommendations and compute their objectives based on the domain-specific initial structure.

During the preprocessing phase, the weight combinations are generated, utility computations and optimizations are performed, and an initial database structure is constructed containing all possible feasible recommendations for the given problem.

Next, the initial database entries are unified, and the Pareto preprocessing database is generated containing the recommendations for optimal trade-offs between competing objectives. These activities serve as the building blocks for establishing the system's base, ensuring a successful execution of the runtime phase.

The runtime phase involves the implementation of the system's functionality, including preparing Pareto-optimal alternatives for the Pareto front, generating the user interface, and handling user interface interaction. This phase is critical in ensuring the system's effective operation and the satisfaction of user needs.

In summary, the two primary phases provide a structured approach to the successful development and operation of the system, from its foundation to its execution. The presented flowchart in Figure 8 illustrates the architecture of the *OptiGuide* system.

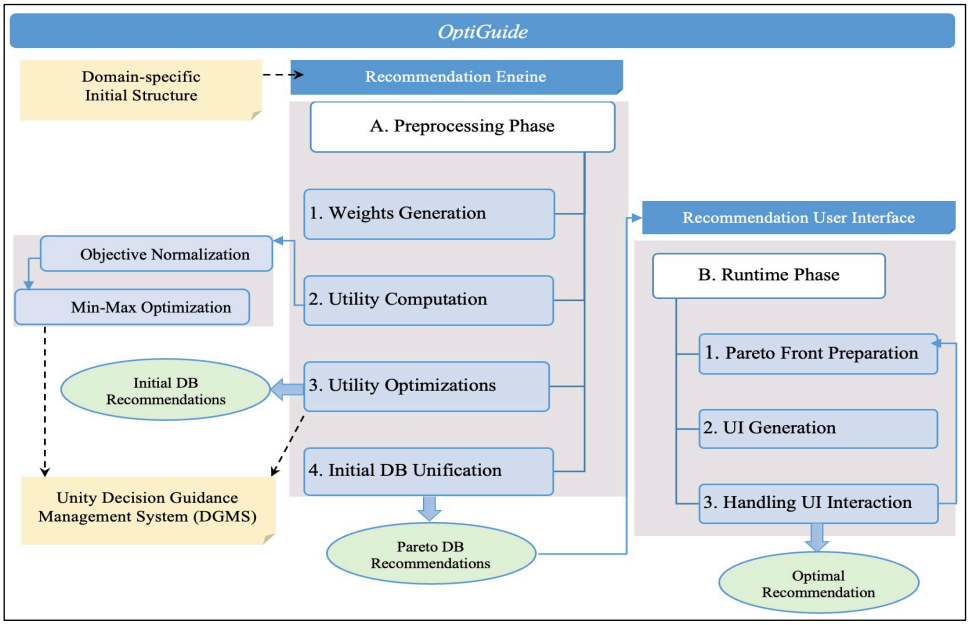


Figure 8. System architecture.

4. System implementation

OptiGuide is implemented using the Python programming language. In the development of the recommendation engine system, Unity DGMS is employed as a platform for solving optimization problems and generating the optimal recommendations using the Decision Guidance Analytics Language (DGAL). Several Python libraries including NumPy, Pandas, and Scikit-learn are used for data processing and analysis. For the recommendation user interface, "Python bindings for the Qt cross-platform application toolkit" (PyQt5) is used for building the graphical user interface (Limited, 2023). Matplotlib, a Python library for creating interactive visualizations, is used to enhance the visual presentation of data. The two following sections briefly explain the implementation aspects pertaining to each component shown in Figure 8 of the *OptiGuide* system.

4.1. Preprocessing Phase

The goal of this phase is to generate a preprocessing structure for Pareto recommendations to be used in the runtime phase. As shown in Figure 8, this phase consists of four steps: Weights Generation, Utility Computation, Utility Optimizations, and Initial DB Unification.

1. Weights Generation. This component generates a list of weight combinations represented as weight vectors, where each weight is associated with an objective. The function takes as input a dictionary of the defined objectives and the number of weight entries. It first computes a "delta" value, which determines the spacing between the weight values, and it finds a list of "alphas" values that are spaced evenly within the specified range of delta. Next, the sine values of alphas are computed to get the initial weights. Using sine values is a technique to create a diverse set of weight combinations, which leads to various trade-offs among objectives. The function then checks the validity of each weight combination by ensuring that the square root of the summation of the squares of the weights is equal to one. The result is a list of weight combinations that will be used in the utility optimization to generate the Pareto-optimal solutions. Consider the case of three objectives: accuracy, cost, and time. The function computes weight combinations using sine values within a specified range. For instance, the weight vector {"accuracy": 0.52, "cost": 0.25, "time": 0.82} might be generated. To ensure validity, the function confirms that the square root of the sum of squares of weights equals 1. These weight vectors contribute to generating Pareto-optimal solutions during optimization.

2. Utility Computation. The idea behind using the utility function is to provide a way to evaluate and compare different objectives in a unified manner, considering the different weights assigned to the objectives. The utility function normalizes the objectives and then computes the weighted sum of the normalized objectives using the generated weight vectors. The result is returned as the utility value of the objectives.

- **Objective Normalization.** This function is used to normalize the objectives within a scale of zero to one, where zero represents the worst value and one is the best. The normalization process considers whether the objective is a minimizing or a maximizing metric.
- **Min-Max Optimization.** This function employs Unity DGMS in order to optimize each objective and find its minimum and maximum values. The difference in constraints for minimization and maximization metrics lies in the bounds applied to the objective value during optimization. These minimum and maximum values are used for the process of normalizing the objectives.

3. Utility Optimizations. This component constructs an "initial DB" list that contains all possible feasible recommendations. The function iterates over the generated list of weight combinations and calculates the utility for each weight combination. Then, it optimizes the computed utility using "dgal.max" function in Unity DGMS along with the given analytic model, variable input, and specified constraints. Once the optimal input and output are found, the objectives and normalized objectives are computed, and the recommendation structure is formed and appended to the "initial DB" list.

4. Initial DB Unification. When generating a list of recommendations, it is common to have some of them with similar or identical objective values but different weights. This can lead to redundancy, making it difficult for a user or decision-maker to choose among them. To address this issue, we implement this function in order to combine similar objective dictionaries in groups using Euclidean distance and a specified maximum distance value. Next, the representative weight dictionary is identified for each group using K-Medoids clustering. The resulting "Pareto DB" is a JSON file that contains only representative entries, which cover all Pareto-optimal recommendations that are non-dominated solutions. This reduces the number of recommendations without losing important information.

4.2. Runtime Phase

The runtime phase of the system aims to facilitate user interaction and learning of an individual's utility to eventually find their optimal recommendation. This phase comprises three steps, namely Pareto Front Preparation, UI Generation, and Handling UI Interaction, as depicted in Figure 8.

1. Pareto Front Preparation. The purpose of this function is to prepare the Pareto front using the "Pareto DB" entries based on the selected x_axis and y_axis objectives and the current weights. The function starts with computing the current utility for each entry in "Pareto DB" based on the given current weights. Then, it creates a data frame of the graph points with the x_axis and y_axis as columns. While the solutions in the "Pareto DB" file are Pareto-optimal in the original space, projecting them onto two objectives or utility versus an objective can result in the loss of Pareto optimality. This occurs because the projection removes information about the remaining objectives, causing the optimality with respect to the projected objectives to no longer hold. To address this issue, the function applies a mask for the created data frame to filter out the set of non-dominated solutions for the projected space from the set of feasible solutions.

2. UI Generation. This component provides the user with a visual representation of Pareto-optimal solutions. The system's graphical user interface (GUI) is defined by a Python class, which has an initializer method that takes Pareto front data as input. The class plots the initial graph of the Pareto front and populates the table with the corresponding data.

3. Handling UI Interaction. This component handles all interactions within the user interface. For instance, when the user clicks on the "Best" button in the table, the "button_clicked()" method handles the click event. It retrieves the weights of the corresponding Pareto-optimal point, prompts user confirmation, calls "paretoOptimal()" and "update_state()" methods to update the GUI with the new data, and finally adds the selected point to the Best-So-Far structure.

5. Conclusions

In this paper, we introduced *OptiGuide*, a domain-independent package recommender system that uses efficient multi-objective optimization techniques to guide users effectively in finding their Pareto-optimal recommendations. The system improves upon earlier research by incorporating preprocessing algorithms to reduce the processing time required for the optimization computations which can enhance the efficiency of the system. The user is effectively engaged in the decision-making process and is provided with ease of UI navigation along with several customization options. Moreover, by implementing a dynamic configuration mechanism, the system becomes more generic and applicable across different domains.

Overall, the proposed system offers a promising approach to generating package recommendations and has the potential to improve decision-making processes in a variety of domains. In future work, we will enhance the user interface options to increase the usability and accessibility of the system. Further experimental studies will be provided as well using real-world applications from different domains.

6. References

Brodsky, A., & Luo, J. (2015). Decision guidance analytics language (Dgal)—Toward reusable knowledge base centric modeling: *Proceedings of the 17th International Conference on Enterprise Information Systems*, 67–78.

Dacrema, M. F., Cantador, I., Fernández-Tobías, I., Berkovsky, S., & Cremonesi, P. (2022). Design and evaluation of cross-domain recommender systems. In F. Ricci, L. Rokach, & B. Shapira (Eds.), *Recommender Systems Handbook* (pp. 485–516). Springer US.

Deng, T., Fan, W., & Geerts, F. (2012). On the complexity of package recommendation problems. *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, 261–272.

Dooms, S., De Pessemier, T., & Martens, L. (2014). Improving IMDb movie recommendations with interactive settings and filters. *Poster Proceedings of the 8th ACM Conference on Recommender Systems*, *1247*.

Fayyaz, Z., Ebrahimian, M., Nawara, D., Ibrahim, A., & Kashef, R. (2020). Recommendation systems: Algorithms, challenges, metrics, and business opportunities. *Applied Sciences*, *10*(21), 7748.

Ge, Y., Zhao, X., Yu, L., Paul, S., Hu, D., Hsieh, C.-C., & Zhang, Y. (2022). Toward pareto efficient fairness-utility trade-off in recommendation through reinforcement learning. *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, 316–324.

Harper, F. M., Xu, F., Kaur, H., Condiff, K., Chang, S., & Terveen, L. (2015). Putting users in control of their recommendations. *Proceedings of the 9th ACM Conference on Recommender Systems*, 3–10.

Hti, R., & Desarkar, M. S. (2018). Personalized tourist package recommendation using graph based approach. *Adjunct Publication of the 26th Conference on User Modeling, Adaptation and Personalization*, 257–262.

Jeffries, W., & Brodsky, A. (2017). Composite alternative pareto optimal recommender system (Capors): *Proceedings of the 19th International Conference on Enterprise Information Systems*, 496–503.

Jeffries, W., & Brodsky, A. (2018). Composite alternative pareto optimal recommendation system with individual

utility extraction (CAPORS-IUX): *Proceedings of the 20th International Conference on Enterprise Information Systems*, 328–335.

Kouris, P., Varlamis, I., Alexandridis, G., & Stafylopatis, A. (2020). A versatile package recommendation framework aiming at preference score maximization. *Evolving Systems*, *11*(3), 423–441.

Limited, R. C. (2023, February 6). *PyQt5: Python bindings for the Qt cross platform application toolkit*. https://www.riverbankcomputing.com/software/pyqt/

Nachawati, M. O., Brodsky, A., & Luo, J. (2017). Unity decision guidance management system: Analytics engine and reusable model repository: *Proceedings of the 19th International Conference on Enterprise Information Systems*, 312–323.

Rahaman, M. S., Shao, W., Salim, F. D., Turky, A., Song, A., Chan, J., Jiang, J., & Bradbrook, D. (2021). Moparker: Multi-objective parking recommendation. *33rd International Conference on Scientific and Statistical Database Management*, 237–242.

Wibowo, A. T., Siddharthan, A., Masthoff, J., & Lin, C. (2018). Incorporating constraints into matrix factorization for clothes package recommendation. *Proceedings of the 26th Conference on User Modeling, Adaptation and Personalization*, 111–119.

Zhang, J., Li, M., Liu, W., Lauria, S., & Liu, X. (2022). Many-objective optimization meets recommendation systems: A food recommendation scenario. *Neurocomputing*, *503*, 109–117.

Zheng, Y., & Wang, D. (2022). A survey of recommender systems with multi-objective optimization. *Neurocomputing*, *474*, 141–153.